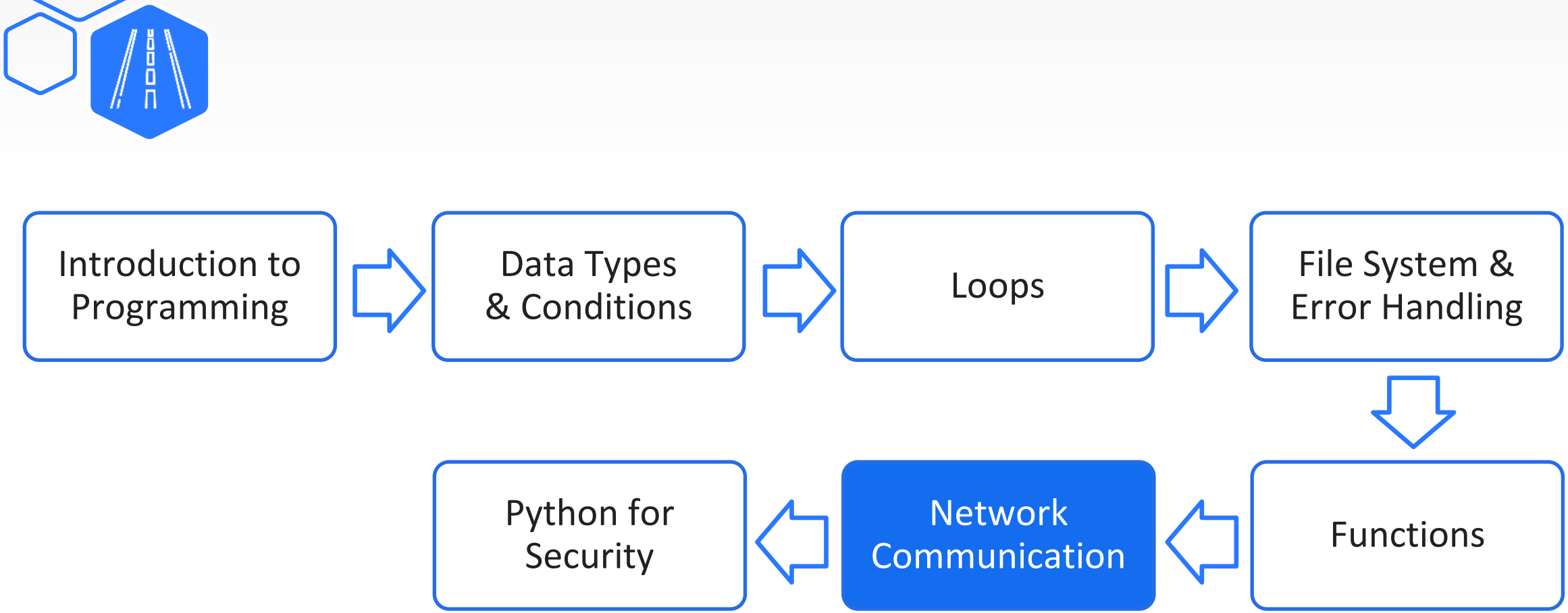
Cybersecurity Professional Program

Network Communication

Introduction to Python for Security

Course Path


Introduction to Programming
Data Types & Conditions
Loops
File System & Error Handling
Functions
Network Communication
Python for Security

Objectives

This lesson focuses on how Python is used for network communication, how sockets are created via Python, and how Python facilitates communication with servers.

- Creating a Server Socket
- Creating a Client Socket
- Sending Data
- Echo Communication

Network Communication

Creating a Server Socket

Sockets

- The endpoint of a connection between two communicating parties
- Consists of an IP address, communication protocol, and port

Import Socket Library

- The socket library is part of Python's standard library.
- The library allows creation of sockets for connection and listening purposes.
- The library includes many functions and variables.

```
import socket
```

Creating a Socket – Server Side

To create a new socket, the socket library must first be imported.

The library includes a method called **socket** that accepts two variables.

```
import socket

new_socket = socket.socket()
```

Setting a Server IP Address and Port

- The socket requires binding to an IP and port.
- The information is passed as a tuple.
- Without the binding, the socket will not be able to accept connections.

```
import socket

new_socket = socket.socket()

new_socket.bind(("0.0.0.0", 50000))
```

Server Connection Limit

- A connection amount limitation can be set.
- The number of connections is represented by an integer.
- The amount of allowed connections must be carefully considered.

```
import socket

new_socket = socket.socket()

new_socket.bind(("0.0.0.0", 50000))

new_socket.listen(4)
```

Accepting Connections

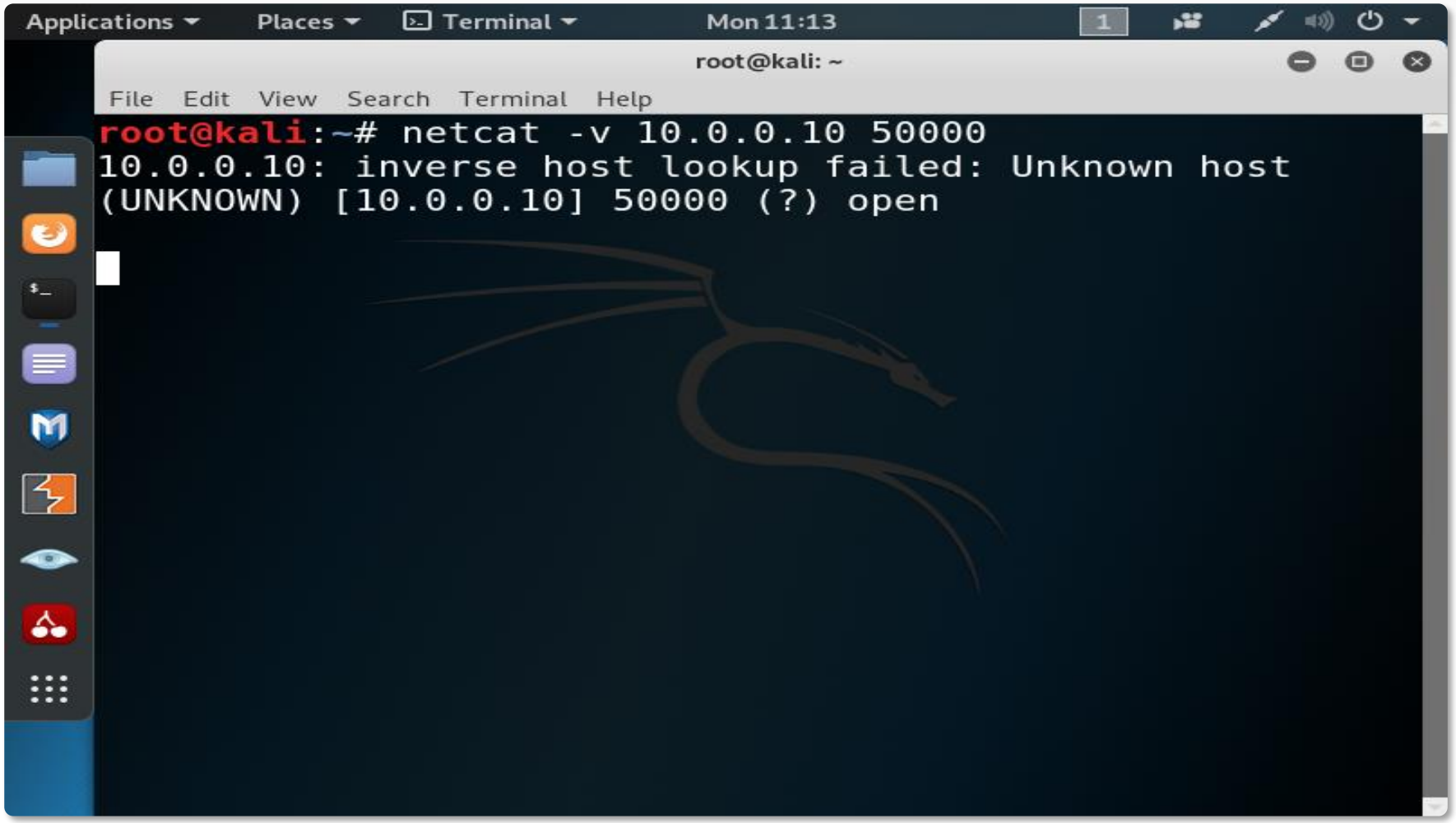
- Two objects are returned when a connection is accepted: a socket and an IP address.
- A connection should be closed when the communication ends.

```
import socket

new_socket = socket.socket()

new_socket.bind(("0.0.0.0", 50000))

new_socket.listen(4)

conn, addr = new_socket.accept()

new_socket.close()
```

Creating a Server Socket

Connection Check

- Each new socket should be tested.
- The **netcat** tool can be used to test the connection.
- The tool acts as a client connected to the server.



Network Communication

Creating a Client Socket

Creating a Socket: Client Side

- The first steps are similar to the server side.
- The library is imported, and the object is created.

```
import socket

new_socket = socket.socket()
```

Establishing a Connection

- Next, the client needs to establish a connection.
- The connection requires parameters similar to the binding.
- Just like with the server side, the connection must be closed when communication ends.

```
import socket

new_socket = socket.socket()

new_socket.connect(("10.0.0.5", 50000))

new_socket.close()
```

Connection Error Handling

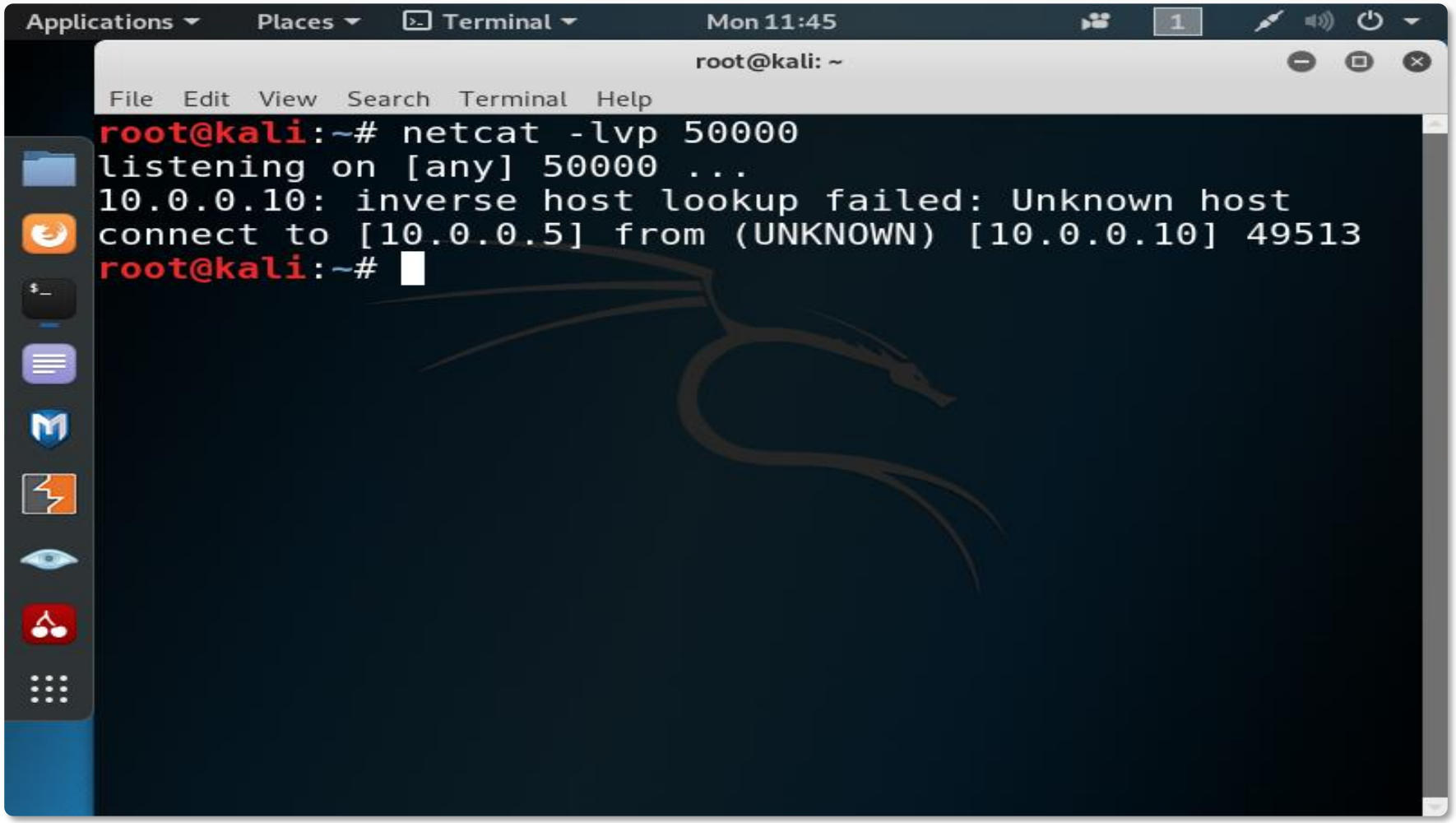
- A client cannot connect to a non-existing server.
- If it tries, an exception will be raised, and the program will be terminated.
- The error can be handled by a ***try & except*** block.

```
import socket

new_socket = socket.socket()

try:
    new_socket.connect(("10.0.0.5", 50000))

except:
    print("The connection was refused")
new_socket.close()
```

Client to netcat: netcat Side

netcat can also act as a server to test clients.

It is configured to listen to incoming connections.



Creating a Client Socket

Lab PY-06-L1

What Are Sockets?

30–45 Min.

Mission

Implement the required commands to create a client connection to a listening server.

Steps

- Sign into the **TDX Arena** platform.
- Navigate to the **Practice Arena**.
- Navigate to the **Python Programming** course.
- Select the ***PY04 Network Programming*** module.
- Select the ***What Are Sockets?*** lab.

Environment & Tools

- Computer
- Internet connection
- Web browser
- TDX Arena access

Related Files

Python Course Textbook
Chapter 6, Section 1 & 2

Network Communication

Sending Data

Sending Data

How Data Is Sent

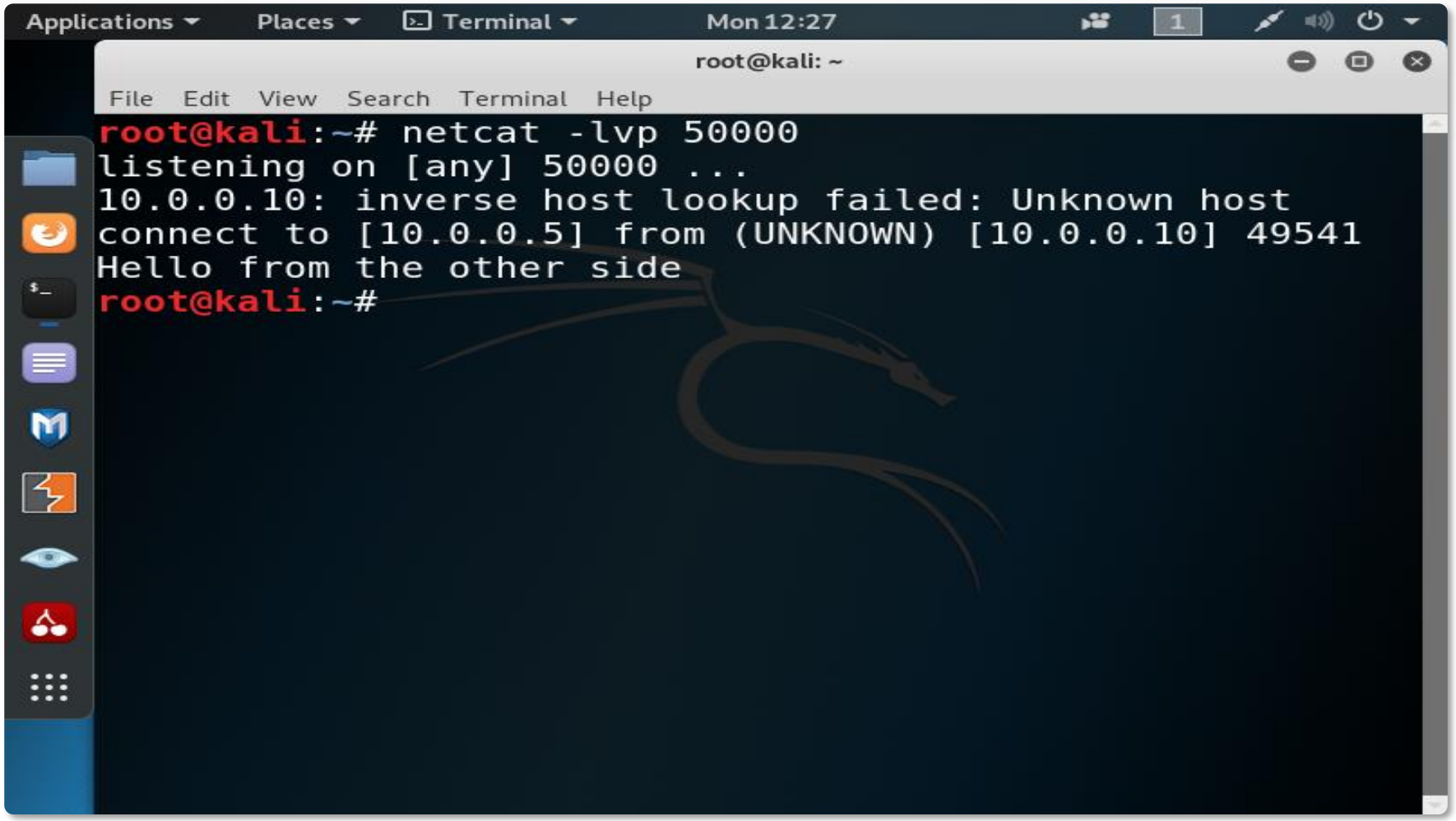
- Data sent between a client and server must be encoded.
- Encoding is required because a socket can only accept bytes.

```
import socket

new_socket = socket.socket()

new_socket.connect(("10.0.0.5", 50000))

new_socket.send("Hello from the other side".encode())

new_socket.close()
```

Sending Data to netcat

The **netcat** server decodes the message.

netcat accepts a connection and then prints the client's message.



Receiving Data: Server Side

- Python's socket can also accept data from a client.
- A byte limitation (buffer) needs to be set for all data received.
- The accepted data must be decoded to get the human-readable representation.

```
import socket

new_socket = socket.socket()
new_socket.bind(("0.0.0.0", 50000))
new_socket.listen(4)
conn, addr = new_socket.accept()
data = conn.recv(2048).decode()
print(data)
new_socket.close()

------------------------------------------------------------
Hello from the other side

Process finished with exit code 0
```

Receiving Data: Client Side

A server typically also sends data to clients.

Clients receive the data similarly to how the server receives data.

```
import socket

new_socket = socket.socket()
new_socket.connect((“10.0.0.5", 50000))
data = new_socket.recv(2048).decode()
print(data)
new_socket.close()

----------------------------------------------------------
Hello from server

Process finished with exit code 0
```

Sending Large Data

- Data sent to the server is limited to 2,048 bytes.
- If a message is more than 2,048 bytes, it will be divided into smaller parts.
- A packet length of less than 2,048 means all the data was transferred.

```
import socket
try:
    mysocket = socket.socket()
    mysocket.bind(("0.0.0.0", 1334))
    mysocket.listen(1)
    c, addr = mysocket.accept()
    buffer = 5
    data = ""
    while True:
          packet = c.recv(buffer)
          parsed = packet.decode()
          data += parsed
          if len(packet) < buffer:
              print("All the data has been received successfully.")
              print("Full data -> {}".format(data))
              break
```

```
<Output in here>

All the data has been received successfully.
Full data -> Hello, I'm the client.

Process finished with exit code 0
```

Sending Data

Lab PY-06-L2

Network Protocol Communications

15–25 Min.

Mission

Write code that will communicate with a server to exchange data.

Steps

- Sign into the **TDX Arena** platform.
- Navigate to the **Practice Arena**.
- Navigate to the **Python Programming** course.
- Select ***PY04 Network Programming***.
- Select the ***Network Protocol Communications*** lab.

Environment & Tools

- Computer
- Internet connection
- Web browser
- TDX Arena access

Related Files

Python Course Textbook
Chapter 6, Section 3

Network Communication

Echo Communication

Echo Server Communication

A *chat* can be established between the client and server.

while true will run as long as a connection is active.

The response will be displayed on the console.

```
import socket
try:
    mysocket = socket.socket()
    mysocket.bind(('0.0.0.0', 50000))
    mysocket.listen(1)
    print("waiting for connection...")
    c, addr = mysocket.accept()
    print("client that connected details are -> {}".format(addr))

    while True:
        sendData = input("message to client :  ")
        c.send(sendData.encode())
        if sendData == "exit":
            c.close()
            break

        rcvData = c.recv(1024).decode()
        print("message from client :  {}".format(rcvData))

        if rcvData == "exit":
            print("connection as closed by user".encode())
            c.close()
            break

except Exception as e:
    print(e)
```

Echo Client Communication

- The client connects to the IP address of the server.
- When a connection is established, the server sends data to the client.
- When the client gets the data, it sends data back to the server.

```
import socket
try:
    mysocket = socket.socket()
    mysocket.connect(('127.0.0.1', 50000))
    print("connection establish...")

    while True:
        serverData = mysocket.recv(2048).decode()
        print("message from server : {}".format(serverData))

        if serverData == "exit":
            print("connection as closed by server")
            mysocket.close()
            break

        sendData = input("message to server : ")
        mysocket.send(sendData.encode())
        if sendData == "exit":
            print("connection as closed by client")
            mysocket.close()
            break

except Exception as e:
    print(e)
```

Echo Communication

TDX Arena Challenge

Fireflies

Mission

> Study the logic of the site and automate the process.

Steps

- Sign into the **TDX Arena** platform.
- Navigate to the **Challenges Arena**.
- Under **Explore More**, select the *Search* icon.
- In the search field, type **fire**.
- Select the ***Fireflies*** challenge.

Then, complete the steps below:

- Understand the site logic.
- Write a script to automate the process.

Thank You

Questions?